Ruby - Bug #18474

938e027c seems to have caused a regression in yield handling with concurrent-ruby

01/11/2022 04:06 PM - aaronjensen (Aaron Jensen)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	3.1.0	**Backport:**	2.6: REQUIRED, 2.7: REQUIRED, 3.0: REQUIRED, 3.1: DONTNEED

Description

I'm sorry for the awful title, I don't know enough about what is going on to describe it better. From a debugging perspective it seems that a very specific set of circumstances causes a begin block to be jumped out of when a proc it is calling yield, without anything being rescuable. An ensure block will be evaluated, if added, but nothing after that. This is begin block: https://github.com/ruby-concurrency/concurrent-ruby/blob/52c08fca13cc3811673ea2f6fdb244a0e42e0ebe/lib/concurrent-ruby/concurrent/executor/safe_task_executor.rb#L23-L29

For reference, see this issue: https://github.com/ruby-concurrency/concurrent-ruby/issues/931

I have a repro here: https://github.com/aaronjensen/concurrent-repro

It requires concurrent-ruby. Someone better versed than me may be able to create a repro without it, but there is a lot going on in concurrent-ruby.

Everything in the repro seems to be required, to_a.first works, but first does not for example.

Related issues:

Related to Ruby - Feature #17613: Eliminate useless catch tables and nops fro...	**Closed**
Related to Ruby - Bug #18475: Yielding an element for Enumerator in another t...	**Closed**
Related to Ruby - Bug #18637: Segmentation fault for yield inside another Thread	**Closed**
Related to Ruby - Bug #18649: Enumerable#first breaks out of the incorect blo...	**Closed**

History

#1 - 01/11/2022 04:45 PM - mame (Yusuke Endoh)

- Related to Feature #17613: Eliminate useless catch tables and nops from lambdas added

#2 - 01/11/2022 06:30 PM - mame (Yusuke Endoh)

Thank you for your report.

I don't yet know exactly what's going on, but as far as I see your repro code, the behavior of yield "some-value" is different from 1.times { yield "some-value" } on Ruby 3.0, and they are the same on Ruby 3.1. I have a feeling that Ruby 3.0 is buggy. I'll take a better look later.

#3 - 01/12/2022 07:27 AM - mame (Yusuke Endoh)

- Related to Bug #18475: Yielding an element for Enumerator in another thread dumps core added

#4 - 01/12/2022 08:21 AM - mame (Yusuke Endoh)

I think I understand the issue. When the following code is executed on Ruby 3.0, it prints "should_not_reach_here".

```
def foo
  Thread.new do
    1.times do
      yield
    end
    p "should_not_reach_here"
  end.join
end

to_enum(:foo).first #=> "should_not_reach_here"
```

This behavior is obviously wrong. Because Enumerable#first does break internally, it must not reach the line of p "should_not_reach_here". Instead, it should raise LocalJumpError or something like this.

```
def foo
  Thread.new do
    1.times do
      yield
    end
    p "should_not_reach_here"
  end.join
end

foo { break } #=> break from proc-closure (LocalJumpError)
```

938e027cdf019ff2cb6ee8a7229e6d9a4d8fc953 fixed this wrong behavior (whether the author @tenderlovemaking (Aaron Patterson) intended to or not).

It is unfortunate that the fix affected concurrent-ruby, but I think this fix should be backported to older ruby branches. I am sorry but I don't know how we can fix concurrent-ruby. I don't know whether it is possible to catch non-local exit event (such as break) in a block that a method yields to.

BTW, there is another (more critical) issue. When the code in question is executed on Ruby 3.1, it dumps core.

```
def foo
  Thread.new do
    1.times do
      yield
    end
    p "should_not_reach_here"
  end.join
end

to_enum(:foo).first
#=> #<Thread:0x00007fa51de97108 -:2 run> terminated with exception (report_on_exception is true):
#   [BUG] Segmentation fault at 0x0000000000000018
```

Ruby 3.0 also dumps core if 1.times do is removed.

```
def foo
  Thread.new do
    yield
    p "should_not_reach_here"
  end.join
end

to_enum(:foo).first
#=> #<Thread:0x000055cfd9e60d58 -:2 run> terminated with exception (report_on_exception is true):
#   [BUG] Segmentation fault at 0x0000000000000018
```

First I thought this is the same issue, but it is different from this ticket. I'll discuss this in #18475.

#5 - 01/13/2022 01:12 AM - aaronjensen (Aaron Jensen)

Thank you, this is helpful. I'm not sure whether or not concurrent-ruby is the right place to fix this. If someone were doing what your example does in a loop and then joining all of them and making that enumerable, it would be their responsibility to ensure it actually behaves like an enumerable. It may be that the right thing to do is to add `rescue LocalJumpError` this line. In the case of a local jump all of the promises would still be waited for, which is unfortunate but probably doesn't matter in this specific instance.

The thing that I still do not understand is what it is about concurrent ruby's Promise#wait that allows it to still evaluate the code that is throwing the error. If I understood that better it might point to concurrent-ruby being responsible for respecting LocalJumpErrors.

#6 - 03/16/2022 01:01 PM - Eregon (Benoit Daloze)

- Related to Bug #18637: Segmentation fault for yield inside another Thread added

#7 - 03/18/2022 03:55 PM - Eregon (Benoit Daloze)

- Backport changed from 2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN to 2.6: REQUIRED, 2.7: REQUIRED, 3.0: REQUIRED, 3.1: DONTNEED

@mame (Yusuke Endoh) Thanks for the repro and explanation.
I've been looking into this and summarized some thoughts in
https://github.com/ruby-concurrency/concurrent-ruby/issues/931#issuecomment-1072542582

Since this bug is fixed on 3.1, but 2.6-3.0 all print "should_not_reach_here", I'll close this issue to mark it ready for backport.
This feels like a serious semantic issue (it breaks to the wrong call-with-literal-block), so I think it is important to backport.

#8 - 03/18/2022 03:55 PM - Eregon (Benoit Daloze)

- Status changed from Open to Closed

#9 - 03/18/2022 04:21 PM - Eregon (Benoit Daloze)

- Related to Bug #18649: Enumerable#first breaks out of the incorect block when across threads added

#10 - 03/18/2022 04:21 PM - Eregon (Benoit Daloze)

Actually there is another probably related bug on 3.1: #18649